Iterative Path Optimization for Practical Robot Planning

Andrew Richardson   Edwin Olson

***Abstract*— We present a hybrid path planner that combines two common methods for robotic planning: a Dijkstra graph search for the minimum distance path through the configuration space and an optimization scheme to iteratively improve grid-based paths. Our formulation is novel because we first commit to the minimum distance path, then explicitly relax the path to maximize the clearance up to a user-specified bound. Notably, this formulation yields more predictable paths than potential field methods which try to trade increases in path length for greater clearance around obstacles. These potential field costs infer a trade off that can yield poor paths when the obstacle map is partially observable and has a finite history.**

Some approximations are used to ensure efficient planning, but only a small set of additional behaviors were required to ensure safe operation. Our method has been field tested extensively, as it is the main on-robot path planner for our large team of 14 medium-scale autonomous ground robots and entry to the 2010 Multi Autonomous Ground-robotic International Challenge, MAGIC 2010.

I. Introduction

Our research focuses on large-environment mapping with a team of many autonomous ground robots and a limited amount of remote human guidance. These robots must be capable of robustly traversing indoor and outdoor urban environments. These are environments with no guarantee of a safe solution, incomplete and noisy obstacle maps, and imperfect localization. These issues confound planning and motivate a system which is predictable and explicitly maximizes path safety when planning in near-collision environments.

The classical problem formulation in path planning is a search for the minimum cost path in a graph. Dijkstra's shortest path algorithm and A^* (an extension of Dijkstra's algorithm with admissible heuristics) are reasonable solutions to this generic formulation [4], [9]. When generating topological plans, e.g. urban driving directions, this formulation is straightforward and works well. However, ground robot path planning must safely handle unknown, cluttered, and dynamic environments. Besides minimizing path length (a proxy for travel time), the path planner must account for imperfect vehicle localization and satisfy vehicle kinematic constraints (such as a non-holonomic constraint for vehicles with limited steering angles) or smooth driving needs.

Two major path planning approaches are roadmap and cost-based formulations. Roadmap methods on the Voronoi diagram guarantee maximum clearance, which provides maximal robustness to localization errors [15], [8], [1]. The standard Voronoi diagram, however, generates maximum-clearance paths even in arbitrarily large environments. Some prior works address this by adding in boundaries retracted from obstacles [1]. Other methods, such as Rapidly-exploring Random Trees (RRTs), do not consider clearance and instead randomly generate roadmaps to create paths which satisfy difficult kinematic or dynamic requirements [12]. In contrast, cost-based formulations define a continuous-valued field in which the costs generated around obstacles are determined by artificial-potential functions [10]. However,

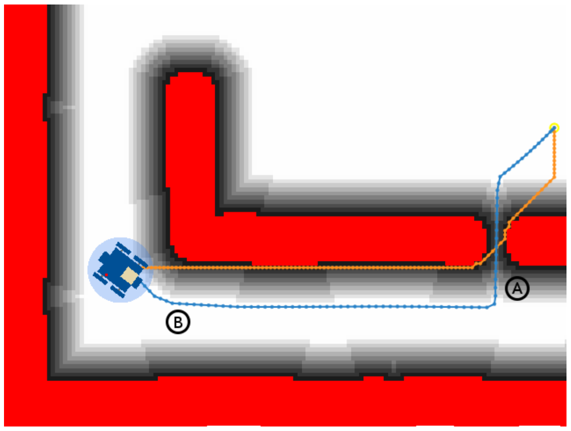


Fig. 1: Example path computed via the proposed method. The configuration space is shown in red, the minimum-distance path from Dijkstra's algorithm in orange, the optimized path in blue, and the distance from the nearest obstacle represented in gray scale. The robot is also drawn in blue. Note that at marker A, the path does not meet the safety margin criteria and is spaced equidistant from both obstacles, while at B the safety margin is met and the path need not extend to be equidistant from obstacles on both sides.

The authors are with the Department of Computer Science and Engineering, University of Michigan, Ann Arbor, USA Email: {chardson,ebolson}@umich.edu URL: http://april.eecs.umich.edu

potential functions can be problematic in varying-size environments because the basic formulation encodes an aversion for small workspaces (a globally-constant function maps distance to cost). Due to this, Dolgov et al. define the Voronoi Field, a potential field scaled by the local workspace size via the Voronoi diagram [5].

Within cost-based formulations, analytic and discrete grid methods are both popular. Analytic methods such as Wein et al. compute closed-form functions for the optimal path given the nearest obstacle, such as a line obstacle [15]. In contrast, grid-based methods represent obstacles and decaying costs with uniformly-spaced discrete samples and use algorithm's such as A* to compute paths [3], [14], [11], [7] . Grid methods also integrate conveniently with terrain classifiers which represent obstacles of arbitrary shapes via a discrete binary obstacle map. These methods are also easy to augment with virtual obstacles from other classifiers. Some grid methods, such as D* and D* Lite, even support efficient replanning when only a portion of the obstacle map changes [14], [11].

Unlike analytical methods, grid methods generally do not create smooth paths. Some methods overcome this issue with line-of-sight checks that remove unnecessary nodes diverging from from the true shortest any-angle path. One example is Theta*, which removes nodes *during* the search on the grid [3]. This closely approximates the any-angle, minimum cost path visibility graph without the high computation time. Of course, the paths will contain some sharp angles, but often fewer than worst-case discrete approximations of diagonal paths. If explicit smoothing is required, the method of Dolgov et al. achieves this through conjugate gradient path smoothing [5]. For efficiency reasons, however, collision-free smoothed paths are not guaranteed without additional analysis after smoothing.

Another strategy for smooth driving is pure pursuit [2]. In pure pursuit, smoothing is achieved in the controller by computing, at every time step, parameters which intersect the vehicle with a look-ahead point down the path. Increasing the look-ahead distance decreases the strictness of the controller and increases the smoothing. This technique results in a smoothed path, but it does not guarantee a collision-free trajectory. Also varying the look-ahead distance as a function of clearance may be sufficient to avoid collisions but does not solve the smoothness problem in narrow environments. Ferguson et al. share this conclusion in their work on Field D*, an interpolation-based planning and replanning algorithm [7].

For many methods, however, an additional problem is caused by the partial-observability of the world state. Even when we retain the last N 3D scans before generating obstacle maps, N must be small to avoid blurring and alignment error. This is problematic for cost-based methods, as an explicit trade off between clearance and distance is defined by the potential function. With a partially-observable world state, there are unobserved states in the map which may deserve a higher cost value than free space. Balancing this cost with the potential functions poses a difficult parameter tuning problem. When tuned poorly, the result is that high costs in the potential field can divert plans through *phantom obstacles*[1] and cause planning oscillations when compared to the minimum distance path.

Our approach is a grid-based planning method using Dijkstra's algorithm. We incorporate obstacle clearance with a cost formulation on a discrete grid, as our terrain classifier outputs discrete, binary-labeled obstacle maps. In addition, we smooth the resulting path from the graph-search to remove high-frequency jitter and ensure that we output a smooth trajectory with meaningful heading references.

We generate plans in a single, 2D configuration space. While our vehicle is not holonomic, its skid-steer drivetrain allows it to turn in place, which we refer to as *quasi-holonomic*. Since the vehicle can turn in place and has a roughly equal aspect ratio, convolution with a single 2D circular kernel efficiently precomputes all collision tests for the vehicle. Our method avoids difficult parameter tuning through a simple heuristic: we first commit to the minimum distance path computed on a binary cost map, then optimize this path to achieve the clearance and smoothness we need.

The contributions of our method are a well tested path planning algorithm for small, quasi-holonomic vehicles that:

1) minimizes path cost in two phases
2) does not require tedious parameter tuning to avoid phantom obstacles and incorporate clearance
3) is largely immune to discrete space planning artifacts
4) ensures that smoothed paths are smoothed *with respect* to the obstacle map
5) is sufficiently fast for real-time use

We describe the method in Section II. In Sections III and IV, we discuss initial plan computation and subsequent optimization routines. In Section V, we provide experimental results using our dataset from the MAGIC 2010 robotics competition.

[1]obstacles which exist but are no longer observed

II. Method

A. Overview

Planning for our robots is split into two problems: *1*) global waypoint planning and *2*) local motion planning. Global waypoint planning is carried out on a remote Ground Control Station (GCS) after the maximum likelihood global map is computed by collectively aligning sub-maps from the robots. As such, global planning is not done on the robot. Instead, the GCS computes coarse global plans and sends a small number of local waypoints (1 to 3) to the robot. In this way, the on-robot planning is indifferent to the larger, multi-robot plan, including the iteratively optimized global coordinate system. On-robot planning is only defined in the local sensor range with limited history.

Motion planning for the robot follows a simple 2D planning scheme based on a discrete cost map. Each discrete cost map is generated from the LIDAR data where the most recent N 3D scans are projected into the XY ground plane and collectively aligned using laser scan-matching.

In this representation, the world is divided into a regular grid of cells in which each cell is labeled as free or infinite cost, the latter of which denotes collisions. Some of our methods could be easily extended for intermediate cost values.

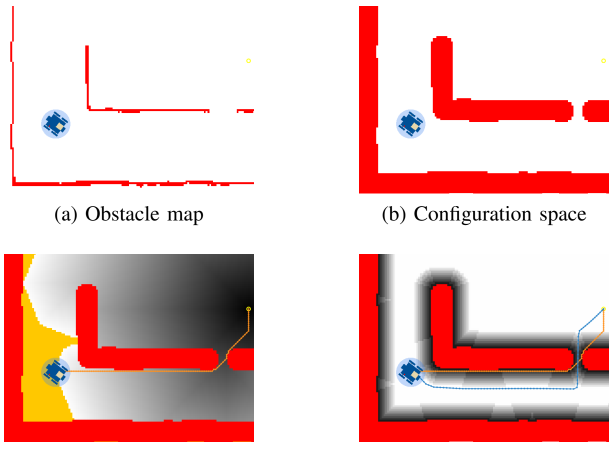
Collision tests are efficiently handled by generating a configuration space through 2D max disc convolution [13]. This encompasses all θ values for a given position and reduces the 3D planning problem to a 2D problem. Further, this reduces the number of points to sample at a given position – only one point must be sampled if the robot is fully covered by a single kernel.

Plan formulation is split into two main steps. First, the minimum distance path is computed by a state space search using Dijkstra's algorithm. Second, this initial path is iteratively optimized to maximize clearance between obstacles when in narrow passageways and smooth the discrete-space path in the continuous domain. This two-stage method was eventually chosen to prevent plans from traversing *phantom openings* hallucinated due to finite sensing history, which can cause planning oscillations.

III. Initial Path: Minimum Distance Plan

A. Configuration Space

As discussed in Section II, collision checking is achieved by pre-computing a configuration space using a circular representation of the robot. The result is a planning space in which collision tests have been precomputed for every state in the state space. Given this



(a) Obstacle map (b) Configuration space (c) Minimum distance path (d) Smoothed result

Fig. 2: Obstacle map and configuration space used for computing the minimum distance path to the goal. The configuration space (b) is computed by 2D max disc convolution with the obstacle map (a). (c) shows the path resulting from Dijkstra's algorithm in orange with the cost of each node examined in gray. Nodes with costs greater than the solution not examined and shown in yellow. Finally, the distance transform (gray) and smoothed path (blue) are shown in (d)

representation, the path planning problem is reduced to a search for a sequence of adjacent states connecting the current robot position to the desired goal. An example input obstacle map is shown in Figure 2a with the corresponding configuration space shown in Figure 2b.

B. Dijkstra's Algorithm

With the configuration space computed, finding the minimum distance path is a simple state space search using Dijkstra's algorithm, sometimes referred to as Wavefront [4]. The algorithm is simple: we initialize the cost of all states to the maximum value and set the goal to zero, adding the goal to a heap. Then we loop, pulling a node from the heap, computing the cost to reach all of its adjacent states in the graph[2], and if that state has not been checked, set its value to the new value and put that state in the heap. This is continued until either the heap is empty or we reach the pose of the robot, thus completing the search. Dijkstra's algorithm has a few nice properties including completeness and optimality, subject to the discretization of the underlying graph. Figure 2c shows the costs computed for the Dijkstra search rendered in grayscale.

[2]We use an 8-connected graph, which has edges at evenly-spaced 45° increments.

Once the costs are computed, the minimum distance path can be computed quickly via gradient descent over the resulting surface, starting at the robot pose and concluding upon reaching the goal.

IV. Path Optimization

With a 5 cm grid and 45° angular increments for planning, the resulting minimum distance path will be coarse. An intuitive but problematic solution is to effectively filter the path by tuning the path controller to follow the path loosely, as discussed in Section I. However, the internal state in the path planner provides useful information for making informed decisions about where to relax the path. Because the 8-connected grid is too coarse to produce a smooth path and a finer grid would increase computational cost, we instead iteratively smooth the path in the continuous domain. The criteria we maximize are *a*) clearance (up to a set bound), and *b*) path smoothness

Each smoothing step requires the calculation of the distance to the nearest obstacle. We precompute this with a distance transform, then iteratively smooth the path for a fixed number of iterations. Additionally, we intend to maximize the distance from the nearest obstacle at all points on the path, up to some maximum clearance.

We are interested in optimizing the following cost function. Let

$$\sum_{i=1}^{N} \delta_l(i)\Delta l_i \tag{1}$$

define the cost computed as a discrete line integral along the path, where $\delta_l(x)$ denotes the sample cost for the point x on the path. We will optimize this function in a decoupled fashion: for each point, minimize the sample cost first, then minimize the local line integral.

This process is implemented as an iterative path optimization, repeating the following three steps for a fixed number of iterations:

1) **Resampling** Ensure minimum and maximum sampling density of the path.
2) **Relaxation** Relax the path by moving points individually down the cost surface to increase clearance, up to a set bound.
3) **Smoothing** Smooth and tighten the path around the cost surface.

A. Distance Transform

We consider the common case where the cost at a point is defined as a function of the distance to the nearest obstacle. This distance transformation is used for relaxation, smoothing, and later to provide clearance information to the path controller for dynamic velocity adjustments. It is computed on the binary configuration space using a method from Felzenszwalb and Huttenlocher and retained only for distances within the clearance bound [6]. An example result is shown in Figure 2d.

Algorithm 1 Path relaxation

```
 1: for $i \in [1, path.length - 1)$ do
 2:     $A = path.get(i-1)$
 3:     $B = path.get(i)$
 4:     $C = path.get(i+1)$
 5:     $best = B$
 6:     for $B' \in SamplesPerpindicularTo(A, C)$ do
 7:         if $cost(B') < cost(best)$ then
 8:             $best = B'$
 9:         end if
10:     end for
11:     $path.set(i, best)$
12: end for
```

B. Path Relaxation to Maximize Safety Margins

Path relaxation is achieved by proposing sideways movements of each point i on the path with respect to the line segment formed by the points $i-1$ and $i+1$. For each point in each iteration, the sample with the minimum cost is taken, even if it would increase the total *integral* path cost in Equation 1. This relaxation step allows the path descend the cost surface, either settling into local minima or stopping once the safety margin has been achieved, and does so by choosing the sideways movement for the point i that minimizes the expression

$$\delta_l(i) \tag{2}$$

independently for each point.

The relaxation algorithm is detailed in Algorithm 1.

C. Path Smoothing

Like path relaxation, path smoothing is achieved by proposing sideways moves for each point i in the path. The difference, however, is that the proposed moves must bring point i closer to the line segment defined by the adjacent points $i-1$ and $i+1$ as well as reduce the total cost of the sub-path $\{i-1, i, i+1\}$. Specifically, we replace the point i with the projection of i onto the line segment $\{i-1, i+1\}$ if the following expression decreases after projection

$$\delta_l(i-1)\Delta l_{i-1} + \delta_l(i)\Delta l_i \tag{3}$$

where Δl_j is the Euclidean distance from point j to $j+1$. This is the local line integral for the triplet.

Computing the total sub-path cost, evaluated as a line integral above, allows us to shorten and thus smooth the path even in uniform-cost regions. This process is outlined in Algorithm 2. Figure 2d shows the relaxed and smoothed path in blue.

Algorithm 2 Path smoothing

```
 1: for $i \in [1, path.length - 1)$ do
 2:     $A = path.get(i-1)$
 3:     $B = path.get(i)$
 4:     $C = path.get(i+1)$
 5:     $seg = LineSegment(A, C)$
 6:     $B' = ProjectionOntoSegment(seg, B)$
 7:     if $pathCost(A, B', C) \leq pathCost(A, B, C)$ then
 8:         $path.set(i, B')$
 9:     end if
10: end for
```

One important note, however, is that we do not run our path smoother until it computes the true shortest diagonal path, which an any-angle planner might compute. While getting rid of unnecessary low frequency artifacts from planning on a regular grid would be beneficial, we are mostly concerned in the high frequency components that cause sensor noise.

D. Configuration Space Approximation and Replanning

The use of a single, rotationally invariant kernel for configuration space generation, as discussed in Section III-A, proved problematic as narrow regions within which the robots could comfortably travel were discarded due to quantization noise and kernel size. For this reason, we reduced the kernel size and set it to the width of the robot, making the configuration space only approximately valid. This works well for our platform because it is only slightly oblong (see Figure 3)

In situations where obstacles are only present on one side of the path, we *guarantee* collision-free performance by iteratively relaxing the path until it is a margin of safety further away from the edge of the configuration space. In situations where the path is instead straddled on both sides by obstacles, the path relaxation allows the path to descend into the local minimum between these obstacles and maximizes the distance from the nearest obstacle on both sides. Both of these scenarios are illustrated in Figure 1.

The most significant remaining failure mode for collisions is a subtle case – replanning events triggered by a change in goal from the GCS may result in a turn in place action that would cause a collision. The preconditions for such a scenario are that that the robot is in a narrow passageway when a new plan is requested and the new destination is behind the robot. This would normally result in an unsafe turn in place action followed by a reasonable path to the goal. If the configuration space kernel fully covered the robot, this action would be safe and correct.

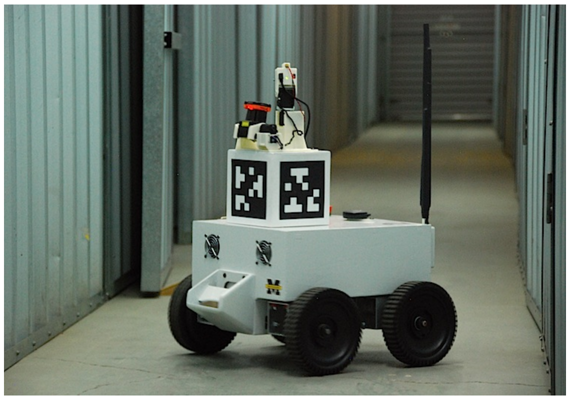


Fig. 3: Each of Team Michigan's 14 autonomous ground robots carry a Hokuyo UTM-30LX laser range-finder and a pan-tilt color camera on a skid-steered drivetrain.

To prevent this situation for the reduced kernel, completed plans are checked for these conditions explicitly. If necessary, the planner will temporarily issue a path to the nearest safe area in front of the robot before honoring the goal from the GCS. The last remaining failure case is a modest turn in a long, narrow environment. We could check for this case, but have not encountered it in practice.

V. Evaluation

Average run-times for path planning on map data from the MAGIC 2010 competition are listed in Table I. In its current configuration, it takes approximately 1.5 s to acquire a full 3D snapshot by tilting a planar LIDAR rangefinger and approximately 0.1-0.2 s to convert the 3D data to an obstacle map. Path planning takes only 0.135 s, completing long before the next 3D scan is fully acquired. The resolution of our obstacle map and configuration space for planning is 5 cm.

All algorithms excluding raw data acquisition are implemented in Java due to its low development overhead and aggressive just-in-time compilation. All run times were measured on a Dell laptop with a 2.53 GHz Intel Core2 Duo CPU.

Additionally, a histogram of the transition angles between adjacent segments in the paths generated during one phase of the MAGIC 2010 robotics competition is shown in Figure 4. It is clear via this plot that after smoothing, the prevalence of sharp angles in the paths is significantly reduced.

Average Run-times over Phases 1 and 2	
Task	Time (s)
2D disc convolution	62.6 ms
Dijkstra's algorithm	6.7 ms
Render distance transform	1.9 ms
Relax and smooth path	20.9 ms
Other	42.5 ms
Total	**134.6 ms**

TABLE I: Run-times for each stage in the proposed algorithm averaged over all 3 Phases of the MAGIC 2010 competition run.

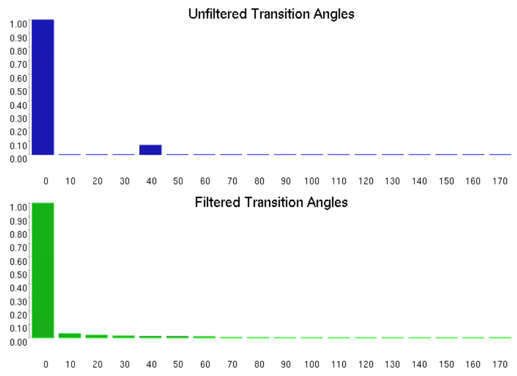


Fig. 4: Histogram of transition angles (in degrees) between adjacent segments in paths generated before and after smoothing. Note the reduction of the mode at 45°, an artifact of 8-connected grid planning, after smoothing.

VI. Summary

We have presented a practical and robust path planning algorithm using a hybrid approach through Dijkstra's algorithm and iterative optimization for bounded clearance maximization and path smoothing. This solution aims to be both computationally efficient and highly reliable. It was heavily tested on a 14-robot system.

It is designed to handle narrow passageways with little more clearance than the width of the robot and does so well, even in the presence of localization and discretization error. This method provides a compelling alternative to artificial potential field approaches with decaying costs around obstacles because clearance is added *after* committing to a path and does not require significant parameter tuning. Additionally, this method will not cause a collision when smoothing the path, and the resulting smooth path contains meaningful heading references for the path controller.

Acknowledgements

This research was supported in part by the Ground Robotics Reliability Center (GRRC) at the University of Michigan, with funding from government contract DoD-DoA W56H2V-04-20001 through the US Army Tank Automotive Research, Development, and Engineering Center.

UNCLASSIFIED: Dist. A Approved for public release.

References

[1] P. Bhattacharya and M. L. Gavrilova. Voronoi diagram in optimal path planning. In *International Symposium on Voronoi Diagrams in Science and Engineering*, pages 38–47, 2007.

[2] R. C. Coulter. Implementation of the pure pursuit tracking algorithm. Technical report, Carnegie-Mellon University, January 1992.

[3] K. Daniel, A. Nash, S. Koenig, and A. Felner. Theta*: Any-Angle Path Planning on Grids. *Journal of Artificial Intelligence Research*, pages 533 –579, 2010.

[4] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959. 10.1007/BF01386390.

[5] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel. Path planning for autonomous driving in unknown environments. *Experimental Robotics*, 54(Figure 1):55–64, 2009.

[6] P. F. Felzenszwalb and D. P. Huttenlocher. Distance transforms of sampled functions. Technical report, Cornell Computing and Information Science, 2004.

[7] D. Ferguson and A. Stentz. Using interpolation to improve path planning: The Field D* algorithm. *Journal of Field Robotics*, 23(2):79–101, 2006.

[8] R. Geraerts and M. H. Overmars. Creating high-quality paths for motion planning. *International Journal of Robotic Research*, 26:845–863, 2007.

[9] P. Hart, N. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100 –107, July 1968.

[10] N. Hogan. Impedance control of industrial robots. *Robotics and Computer-Integrated Manufacturing*, 1(1):97 – 113, 1984.

[11] S. Koenig and M. Likhachev. Fast replanning for navigation in unknown terrain. *IEEE Transactions on Robotics*, 21:354–363, 2005.

[12] S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. *International Journal of Robotics Research*, 20(5):378–400, 2001.

[13] T. Lozano-Perez. Spatial planning: A configuration space approach. *Computers, IEEE Transactions on*, C-32(2):108 –120, Feb. 1983.

[14] A. Stentz. Optimal and efficient path planning for partially-known environments. In *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*, pages 3310 –3317 vol.4, May 1994.

[15] R. Wein, J. P. V. D. Berg, and D. Halperin. Planning high-quality paths and corridors amidst obstacles. *International Journal of Robotic Research*, 27:1213–1231, 2008.